LA-UR-01-1063

Title: Mesh Data Structure Selection for Mesh Generation and FEA Applications

Author(s): R. V. Garimella

Submitted to: International Journal of Numerical Methods in Engineering
v 55, n 4, pp. 451-478, Oct 2002

Mesh Data Structure Selection for Mesh Generation and FEA Applications

Rao V. Garimella
Geoanalysis, Earth and Environmental Sciences,
Los Alamos National Laboratory, Los Alamos, NM 87545.
E-mail: rao@lanl.gov

March 12, 2004

Abstract

The data structure representing a mesh and the operators to create and query such a database play a crucial role in the performance of mesh generation and FE analysis applications. The design of such a database must balance the conflicting requirements of compactness and computational efficiency. In this article, ten different mesh representations are reviewed for linear tetrahedral and hexahedral meshes. A methodology for calculating the storage and computational costs of mesh representations is presented and the ten data structures are analyzed. Also, a system for ranking different data structures based on their computational and storage costs is devised and the various mesh representations are ranked according to this measure.

Keywords: Mesh Generation, Data Structures, Tetrahedral and Hexahedral meshes, grids, triangulation.

1 Introduction

As numerical simulation assumes a larger role in the modern design process, the complexity and detail of the models being analyzed is increasing rapidly. One of the most expensive steps in going from the description of a domain as a geometric model to its analysis using numerical methods like finite element methods continues to be the generation of a mesh for the model. It is also a phase which requires automatic algorithms capable of creating large meshes satisfying multiple criteria in complex domains. Therefore, mesh generation is receiving increased attention in academic and industrial environments. In particular, the data structures for mesh representation strongly influence the performance of applications that create the mesh and also of those that merely use it. Therefore, careful optimization of the mesh representations has a significant impact on many aspects of large scale numerical simulations.

Some mesh generation procedures use simple and very compact data structures borrowed from finite element methods. These data structures often consist of elements and nodes, with elements pointing to their component nodes. However, this data structure quickly becomes expensive and inadequate even for moderately complex mesh generation applications. In fact, most mesh generation algorithms that use a rudimentary element-node data structure also store or dynamically build a considerable amount of auxiliary information [1, 2, 3, 4] such as node-to-node or element-to-element connections. This additional data allows them to retrieve more complex relationships in the mesh such as the elements connected to a node. Not only that, most meshing and analysis procedures that use these rudimentary data structures implicitly recognize and utilize the concept of additional mesh entities such as mesh edges (as a connection between two nodes) and mesh faces (as a facet separating two 3D elements).

To meet the needs of a large range of applications with a single mesh data structure, some researchers have adopted more rich and flexible representations based on the B-rep scheme of representing geometric models [5, 6, 7]. This form of mesh representation consists of topological entities (regions, faces, edges and vertices), links between them (called *adjacencies*) and the geometry underlying the topological entities [8, 9]. Mesh regions in this representation may be tetrahedra (4 faces), hexahedra (6 faces) and triangular prisms (5 faces), mesh faces may be triangles (3 edges) or quadrilaterals (4 edges) and mesh edges can have two and only two vertices. The shape or geometry of the entities is often linear but may be quadratic, cubic or even more general for higher order elements [10, 11]. In addition, a mesh entity may store *classification* information [5, 6, 7] relating the entity to the geometric model entity that it partly or fully discretizes. The data structure stores 1-level upward and downward adjacencies, that is relationships between entities differing by one order in dimensionality. Therefore, vertices point to edges using them, edges point to their vertices and to faces using them, faces point to their edges and to regions using them and regions point to their faces.

These two representations are a subset of an large set of possible data structures for representing meshes. All the entity types may be represented in the data structure or some intermediate types (faces, edges) may be left out. With every set of entities represented in the data structure, there are multiple choices for the adjacency relationships that are explicitly stored. The computational cost and memory usage of the data structure is strongly influenced by the connections that are explicitly stored. If all possible connections between entities are stored in a data structure, then the cost of retrieving any entity relationship is a constant. However, this option tends to use impractically large amounts of memory and for dynamic meshes, the cost of modifying the mesh increases as more types of entity relationships are explicitly stored. Therefore, only a subset of all possible adjacency relationships are typically stored while the others are derived. Of these subsets, some sets of connections require hierarchical traversal of connections to retrieve all possible relationships while others require global searches through the mesh, which is a much more expensive option and is generally not considered in most applications.

To maximize the performance of procedures creating, manipulating and using meshes, it is necessary to analyze the various data structures and carefully choose one or more that offer the best solution. Of the many possible data structures, this article presents the 10 most viable mesh representations for mesh generation and numerical analysis. It also presents a comparative analysis of the computational and storage efficiency of these data structures. Finally, it proposes a method for choosing a mesh representation best suited for an application by using some specific applications as examples.

The next section in this paper describes the general requirements of a mesh database for mesh generation and other applications. Section 3 presents the methods used in the paper for analyzing computational complexity of various operations. Section 4 details the notation used in the rest of the discussion. Section 5 and Section 6 present the storage and computational cost of ten different mesh data structures. Six of these mesh data structures are full data structures; in other words, they have all the four types of mesh entities, regions, faces, edges and vertices. The remaining are reduced representations in which one or more types of entities are not stored but are derived when needed. Such representations save memory but require extra computational effort. A summary of the storage and computational costs of all the mesh representations is presented in Section 7 for tetrahedral and hexahedral meshes. Also, a method is proposed to use the storage and computational costs to derive a "goodness" measure for each data structure. Section 8 presents the conclusions of the paper.

2 Requirements of Mesh Data Structures

The factors that must be considered while designing a mesh data structure for mesh generation and finite element analysis applications are *storage* and *efficiency*.

Minimizing the storage requirements of a data structure is very important in mesh database design. It dictates the maximum number of elements in the mesh that can be created or manipulated for a given domain which can have a direct impact on solution accuracy. The memory usage of a mesh data structure is

dependent on the amount of memory each mesh entity uses, the number of entities in the mesh, the amount of memory each connection uses (which is excluded from the memory usage of the mesh entity itself) and the number of connections that are explicitly stored in the data structure. Figure 1 shows all the topological entities of a mesh (vertex, edge, face and region) and all possible connections between them.

Equally important is the computational efficiency of operations with the mesh representation. Any procedure using the mesh database needs access to mesh adjacency information such as the list of vertices of a region, the list of faces of an edge and so on. In addition to adjacency retrivals, mesh generation applications also invoke operators for creating and updating entities. Therefore, the cost of each of these operations influences the computational cost of using a mesh data structure.

Not all mesh query and creation operators are called equally in any application. Therefore, the *relative call frequency* of the operator, in other words, the relative number of times an operator is called, also influences the overall cost of using a mesh representation. To get an idea of the actual cost of the operators in the application, the cost of each operator must be weighted by the relative number of times it is called. Relative call frequencies of the mesh database operators vary widely with the application using the database. They can be quite different for different mesh generation algorithms and vary even in different phases of the same mesh generation procedure itself. Therefore, to compute an overall computational cost for a mesh data structure, call frequencies for an application must be coarsely estimated from a representative population of tests. The coarseness of the computational cost calculation can be reduced if the call frequency is estimated for each major phase of an application. Since the costs will be different for different phases of the application, it follows that it is beneficial to have the capability to switch between representations at will [12].

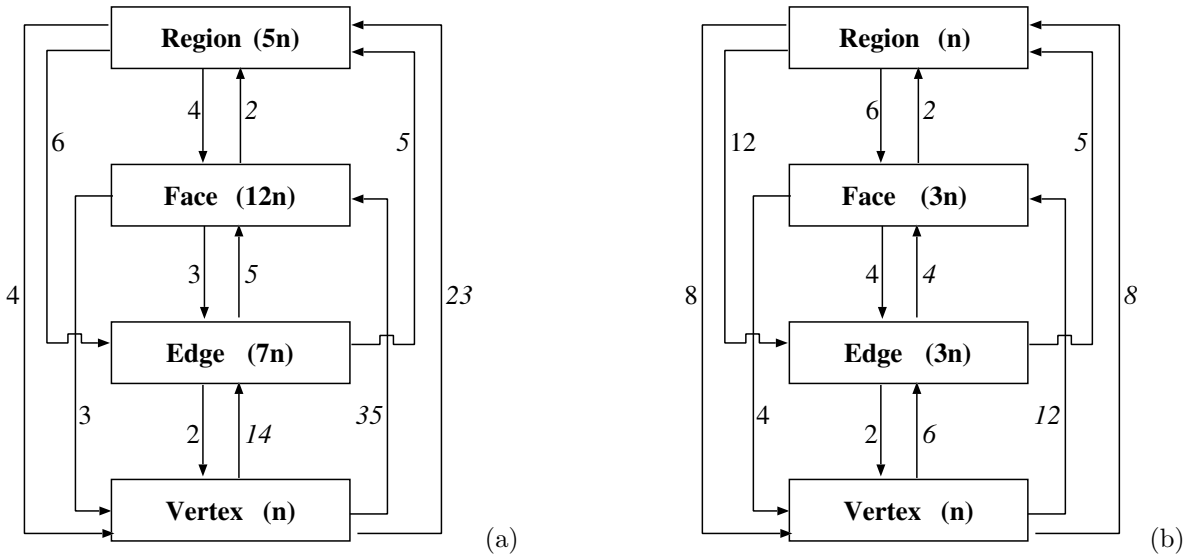

Figure 1: Mesh representation with all entities and connections. Also shown are typical statistics for number of entities (shown in the boxes) and connections (shown next to the links) for a tetrahedral and hexahedral meshes. The numbers for downward adjacencies are exact while the ones for upward adjacencies (in italics) are averages.

3 Analysis of Mesh Data Structures

To estimate the storage requirements of a mesh, a good estimate is needed for the relative numbers of different types of entities in a mesh and the typical sizes of the different upward and downward adjacency sets. Such estimates are presented by Beall, et. al. [7] for tetrahedral and hexahedral meshes and are shown

here in a schematic form in Figures 1a,b. In the figures, the number in parentheses in any box represents the number of entities of that type in a mesh expressed in terms of n, the number of vertices in the mesh. Also, by the side of each connection from entity of type A to entity of type B is the number of entities of type B that are typically connected to an entity of type A. Note that the numbers for upward adjacencies are averages while those for downward adjacencies are exact.

To make the estimation of the memory requirements of the data structures simple, the memory occupied by a mesh entity is accounted for separately from its connections to other entities. Then, the average storage requirement of an entity object can be considered to be M_e and the average storage requirement of a connection can be considered to be M_c. Often, M_c is just the space occupied by a single pointer, while M_e is larger.

Computational efficiency for mesh data structures is analyzed here in terms of operation counts for the various adjacency operations. To do this, the most efficient algorithm is devised for an adjacency retrieval for the particular mesh data structure being analyzed. Then the algorithm is broken down into simple steps like data storage (PUT), retrieval (GET), comparison (CHECK if A equals B) and assignment (ASSIGN). The operation count (*op. count or OC*) for storage, retrieval, assignment and comparison are all taken to be 1 (more precisely, a constant or $O(1)$). For brevity, slightly more complex steps may also be defined as a single operation. For example, checking if an entity E_i belongs to a set $\{E\}$ of size N, involves N comparisons. Hence, its operation count is said to be N.

The calculation of operation counts for adjacency retrieval procedures is illustrated in Alg. 1 using the procedure for getting the faces of a vertex. The data structure used is one where where the upward adjacencies are Vertex-Edge, Edge-Face and Face-Region (e.g. See Figure 8). In a tetrahedral mesh each vertex is connected to an average of $N_1 \approx 14$ edges and $N_2 \approx 35$ faces, and each edge is connected to an average of $N_3 \approx 5$ faces.

```
GET the edges of the vertex                                   /* OC = N1 = 14 */
for (each of the N1 edges) do
  GET the faces of the edge                                   /* OC = N3 = 5 */
  for (each of the N3 faces) do
    CHECK if face is in Faces_Of_Vertex list                  /* OCave = N2/2 ≈ 18 */
    if (face is not in Faces_Of_Vertex list) then
      PUT face in list                                        /* OC = 1 */
    end if
  end for
end for

/* OC = N1 + (N1)(N3 + (N3)(N2/2 + 1)) = (14) + (14)(5 + (5)(18 + 1)) =
1414 */
```

Alg. 1: Getting the faces of a vertex using local searching

In Alg. 1, the first step is a direct retrieval operation and hence the operation count equals the average number of edges connected to a vertex, i.e., 14. For each of these edges, the connected faces (average 5) are retrieved directly. Then a union of the sets of faces connected to the edges is created, denoted by Faces_Of_Vertex. Since each face must occur only once in the final set, one must check for the existence of the face in the set before adding it in. The initial size of the set Faces_Of_Vertex is 0 and its final size after the union operation is complete is an average of 35. Therefore, it is assumed that checking for the existence of a face in the set requires an average of 18 (35/2) comparisons (indicated in algorithms as ***OCave***). Without any further refinements to the algorithm, the total operation count of the procedure is calculated to be 1414.

The process of creating a union of multiple sets of entities, as executed above, is one of the most frequent operations used in mesh adjacency retrieval procedures. Therefore, the improvement of the operation count of a typical algorithm such as Alg. 1 is of value. It is possible to reduce the operation count by stricter

accounting of the various operations. For example, in Alg. 1, the addition of a face to the set of unique entities does not happen $(N_1)(N_3)(1) = (14)(15)(1)$ times. Rather it occurs only $N_2 = 35$ times for the entire process and can be accounted for separately. In the algorithm descriptions, this number is referred to as ***OCtotal***. Therefore, the total operation count may be calculated as OC = 14 + (14)(5 + (5)(18)) + 35 = 1379. Although helpful, this still does not offer a substantial reduction in the computational cost.

A more effective method for reducing the cost of adding entities to sets without duplication is by tagging or marking entities as they are inserted into lists (also see [7]). This eliminates the need for searching through the list to see if an entity is present; instead the entity is checked if it is tagged/marked. The use of marks to track which entities are already in the list reduces the operation cost to a constant instead of depending on the size of the list. In this analysis, the operation count for changing the mark on an entity or checking if the entity is marked is taken as 2 - one for accessing the data field reserved for marks in the entity object and one for changing the value or comparing it to an expected value. The algorithm for retrieving faces of a vertex described before is modified with the use of marks as shown in Alg. 2 and the resulting operation count is reduced to 399.

```
GET the edges of the vertex                                   /* OC = N1 = 14 */
for (each of the N1 edges) do
  GET the faces of the edge                                   /* OC = N3 = 5 */
  for (each of the N3 faces) do
    CHECK if face is marked (is it in Faces_Of_Vertex list?)  /* OCave = 5/2 ≈ 2 */
    if (face is not marked) then

      /* THIS PART EXECUTED 35 TIMES IN TOTAL */

      PUT face in Faces_Of_Vertex list          /* OC = 1, OCtotal = N2 = 35 */
      MARK face (to indicate it is in list)     /* OC = 2, OCtotal = (2)(N2) = 70 */
    end if
  end for
end for

UNMARK faces of Faces_Of_Vertex list                          /* OC = (2)(N2) = 70 */

/* OC = 14 + (14)(5 + (5)(2)) + (35 + 70) + 70 = 399 */
```

Alg. 2: Getting the faces of a vertex using marks

Clearly, the use of marks to keep track of entities in lists offers substantial savings in computational cost over local searching in the example shown. However, for simple adjacency retrievals, such as edges of a region or vertices of a face, the use of marks imposes a larger overhead than using local searching. Therefore, marking is used only in the retrieval of upward adjacencies involving large local searches.

4 Notation

The notation used for the vertices, edges, faces and regions of a mesh are V, E, F and R respectively. $\{F\}$, $\{F_i, i = 1, \ldots, 6\}$, and $\{F_1, F_2, F_3, F_4, F_5, F_6\}$ are all sets of faces. Adjacency relationships are denoted by the () operator. Thus, $F(E)$ denotes a face of an edge. On the other hand, $\{F(E)\}$ denotes the set of faces connected to the edge. $\{F(E), i = 1, \ldots, \approx 5\}$ denotes the same set and indicates that in this case, approximately 5 faces are expected to be connected to the edge. Table 1 shows the notation for all the adjacency operators. Each column in the table represents the adjacency operators for a particular type of entity. The arrow beside each entry indicates if an upward or a downward adjacency is being accessed.

Table 1: Adjacency operators and their notations

	Region	Face	Edge	Vertex
Regions of ()	-	$\{R(F)\} \uparrow$	$\{R(E)\} \uparrow$	$\{R(V)\} \uparrow$
Faces of ()	$\{F(R)\} \downarrow$	-	$\{F(E)\} \uparrow$	$\{F(V)\} \uparrow$
Edges of ()	$\{E(R)\} \downarrow$	$\{E(F)\} \downarrow$	-	$\{E(V)\} \uparrow$
Vertices of ()	$\{V(R)\} \downarrow$	$\{V(F)\} \downarrow$	$\{V(E)\} \downarrow$	-

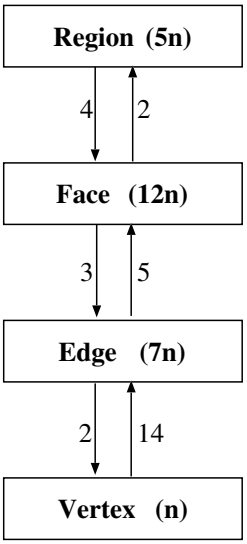

Figure 2: Mesh Representation F1 - Full one-level upward and downward adjacencies.

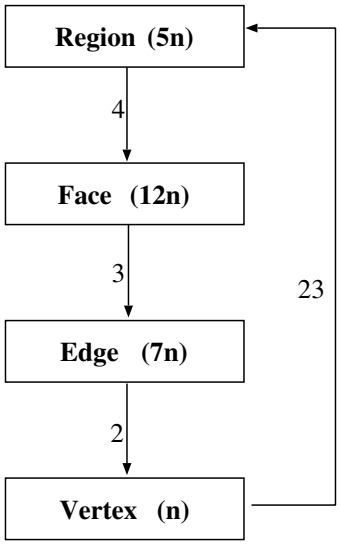

Figure 3: Mesh representation F2 with full downward and vertex-region upward adjacency.

5 Full Mesh Representations

Mesh representations that contain topological entities of all dimensions are referred to here as *full* mesh representations. Full mesh representations always contain vertices, edges, faces and regions (for 3D meshes). However, they need not represent all connections explicitly. The relationship of full mesh representations to a geometric model is unambiguous. Full mesh representations can be used to properly represent mixed dimension meshes, i.e., meshes with a regions, faces with no connected regions, edges with no connected regions or faces and vertices with no connected edges, faces or regions. The storage cost of full representations depends on explicitly represented connections and the computational cost on derived connections.

5.1 Mesh Representation F1

Shown in Figure 2 is a mesh representation [5, 6, 7] with all topological entities and all upward and downward adjacencies of one level (A k-level adjacency is the relationship between entities differing in dimensionality by k). This data structure can represent mixed dimension meshes and its relation to a geometric model is unambiguous.

5.1.1 Storage

As described before, if the number of vertices in a tetrahedral mesh is n, then the approximate number of edge, faces and regions is $7n$, $12n$ and $5n$ respectively. Assuming that each entity uses a memory of M_e on average, the total memory used by all the entities is $(nM_e + 7nM_e + 12nM_e + 5nM_e) = 25nM_e$.

Since there are n vertices and each vertex is used by an average of 14 edges, the memory used by connections from vertices to edges is $14nM_c$, where M_c is the average amount of memory used by each connection. Each of the $7n$ edges of the mesh are connected to 2 vertices and 5 faces. Therefore, connections emanating from the edge use $(7n)(2+5)M_c = 49nM_c$ units of storage. Similarly, connections from faces use $(12n)(3+2)M_c = 60nM_c$ and connections from regions use $(5n)(4)M_c = 20nM_c$ units of storage. Adding all of the above, the total amount of memory used in maintaining all the adjacency information of this mesh representation is $143nM_c$. The total memory usage of this mesh representation is $(25M_e + 143M_c)n$.

5.1.2 Operation Counts

Operation counts for the 12 low level adjacency operators listed earlier are derived here for this mesh representation. Also, operation counts for entity creation operators and if necessary, some higher level adjacency operators used by them are derived. For subsequent data structures, operation count calculations with detailed algorithms are presented in the main text only when deemed necessary. For the rest, the operation count is simply stated in the Summary, Sec. 7. A complete set of algorithms for all the representations are given in [13].

1. $\{F(R)\}$: $OC = 4$, as the faces of a region are directly referenced by the region.
2. $\{E(R)\}$: $OC = 36$; this is a 2-level downward adjacency retrieval whose procedure is described in Alg. 3.
3. $\{V(R)\}$: $OC = 30$; this is a 3-level downward adjacency operation similar to $\{E(R)\}$.
4. $\{R(F)\}$: $OC = 2$, as faces directly reference the regions using them.
5. $\{E(F)\}$: $OC = 3$, as edges of a face are directly referenced by the face.
6. $\{V(F)\}$: $OC = 13$; this is a 2-level downward adjacecny operation similar to $\{E(R)\}$.
7. $\{R(E)\}$: $OC = 50$, this is a 2-level upward adjacency in which it is advantageous to use local searching instead of marks to construct the union of multiple sets of regions (See Alg. 4).
8. $\{F(E)\}$: $OC = 5$, as edges directly reference the faces using them.
9. $\{V(E)\}$: $OC = 2$, as vertices of an edge are directly referenced by the edge.
10. $\{R(V)\}$: $OC = 619$; this is a 3-level upward adjacency in which the marks are used for constructing the union of region sets (See Alg. 5).
11. $\{F(V)\}$: $OC = 399$; this is a 2-level upward adjacency similar to $\{R(V)\}$.
12. $\{E(V)\}$: $OC = 14$, as vertices directly reference the edges using them.
13. $E(V_1, V_2)$: $OC = 70$; given two vertices, this higher level adjacency retrieval operator gets the edge that connects them if one exists. It returns 0 if no such edge exists. This operator is used in procedures to create faces.
14. $F(V_1, V_2, V_3)$: $OC = 155$; given three vertices, this operator retrieves a face connected to all three of them. This operator is used in procedures to create regions.
15. Create V: $OC = 5$.

```
GET F_1(R)                                          /* Get face 1 of region, OC = 1 */
GET {E_i(F_1), i = 1, ..., 3}                       /* Get the 3 edges of face 1, OC = 3 */
PUT the 3 edges in {E(R)}                           /* OC = 3 */

/* Add two edges of face 2 that are not already in the list */
GET F_2(R)                                          /* Get face 2 of region, OC = 1 */
GET {E_i(F_2), i = 1, ..., 3}                       /* Get the 3 edges of face 2, OC = 3 */
for (E_i(F_2), i = 1, ..., 3) do
  CHECK if E_i ⊂ {E(R)}                             /* Check if E_i is in edge set, OC = 2 */
  if (E_i ⊄ {E(R)}) then
    PUT E_i in set {E(R)}                           /* OC = 1, OCtotal = 2 */
  end if
end for

/* Add one edge of face 3 that is not already in the list */
GET F_3(R)                                          /* OC = 1 */
GET {E_i(F_3), i = 1, ..., 3}                       /* OC = 3 */
for (E_i(F_3), i = 1, ..., 3) do
  CHECK if E_i ⊂ {E(R)}                             /* OC = 3 */
  if (E_i ⊄ {E(R)}) then
    PUT E_i in {E(R)}                               /* OC = 1, OCtotal = 1 */
  end if
end for

/* OC = 1 + 3 + 3 + 1 + 3 + (3)(2) + 2 + 1 + 3 + (3)(3) + 1 = 36 */
```

Alg. 3: Mesh Representation F1: $\{E(R)\}$

```
GET F_i(E), i = 1, ..., ≈5                          /* OC = 5 */
for (F_i, i = 1, ..., 5) do
  GET R_1(F_i), R_2(F_i)                            /* OC = 2 */
  for (R_j, j = 1, 2) do
    CHECK if R_j ⊂ {R(E)}                           /* OCave = 5/2 ≈ 3 */
    if (R_j ⊄ {R(E)}) then
      PUT R_j in {R(E)}                             /* OC = 1, OCtotal = 5 */
    end if
  end for
end for

/* OC = 5 + (5)(2 + 2(3)) + 5 = 50 */
```

Alg. 4: Mesh Representation F1: $\{R(E)\}$

```
GET {E_i(V), i = 1, ..., ≈14}                                  /* OC = 14 */
for (E_i, i = 1, ..., 14) do
  GET {F_j(E_i)}, j = 1, ..., ≈5}                              /* OC = 5 */
  for (F_j, j = 1, ..., 5) do
    GET {R_k(F_j), k = 1, 2}                                   /* OC = 2 */
    for (R_k, k = 1, 2) do
      CHECK if R_k is marked                                   /* OC = 2 */
      if (R_k is not marked) then
        PUT R_k in {R(V)}                                      /* OC = 1, OCtotal = 23 */
        MARK R_k                                               /* OC = 2, OCtotal = 46 */
      end if
    end for
  end for
end for
UNMARK regions of {R(V)}                                       /* OC = (23)(2) = 46 */

/* OC = 14 + (14)(5 + 5(2 + (2)(2))) + 23 + 46 + 46 = 619 */
```

Alg. 5: Mesh Representation F1:$\{R(V)\}$

16. Create $E(V_1, V_2)$: $OC = 6$.

17. Create $F(E_1, E_2, E_3)$: $OC = 24$; this is a simple algorithm in which the face object is created and the edges directly incorporated into it. The direction in which the face uses each of the edges is deduced from the ordering of the edges.

18. Create $F(V_1, V_2, V_3)$: $OC = 242$; in this algorithm, the edges of the face must be found (using the $E(V_1, V_2)$ operator) or created from the vertices and then incorporated into the face object (Alg. 6).

19. Create $R(F_i, d_i, i = 1, \ldots, 4)$: $OC = 14$; this is a simple algorithm in which the given faces F_i are incorporated into the region object with the given direction d_i.

20. Create $R(V_1, V_2, V_3, V_4)$: $OC = 1618$, in this algorithm (Alg. 7), the faces of the region must be found (using the $F(V_1, V_2, V_3)$ operator) or created from the vertices and then incorporated into the region. The vertices of each face of the region are deduced from a pre-defined template.

The operation counts for this data structure with the use of marks are summarized in Table 2 and Table 4.

5.2 Mesh Representation F2

This is a full mesh representation containing all the mesh entities but only one set of upward adjacencies, from vertices to regions (Figure 3). To access any upward adjacency other than vertices to regions, it is necessary to travel in a circular fashion. For example, to get the faces connected to an edge, one must get the vertices of the edge, get the common set of regions of the vertices, get the faces of the region and find the faces of the the regions that contain the edge. Therefore, this mesh representation is called a circular representation.

Since this data structure contains all the topological entities, the memory usage by the mesh entities, i.e., $25nM_e$. However, the storage used by the connections is only $((4)(5) + 3(12) + 2(7) + 23)nM_c = 93nM_c$. So the total memory usage by this data structure is $(25M_e + 93M_c)n$.

The operation counts for the data structure are shown in Tables 2 and 4.

```
CREATE face object F                                          /* OC = 1 */
PUT F in the mesh face list                                   /* OC = 1 */
for (i = 1, ..., 3) do
  GET E_i(V_i, V_(i+1)%3)                                     /* OC = 70 */
  if (E_i(V_i, V_(i+1)%3) = 0) then
    CREATE E_i(V_i, V_(i+1)%3)                                /* OC = 6 */
    ASSIGN d_i ← 1                                            /* OC = 1 */
  else
    GET V'_1(E_i)                                             /* OC = 1 */
    CHECK if V'_1(E_i) == V_i                                 /* OC = 1 */
    if (V'_1(E_i) == V_i) then
      ASSIGN d_i ← 1                                          /* OC = 1 */
    else
      ASSIGN d_i ← 0                                          /* OC = 1 */
    end if
  end if
  PUT E_i in {E(F)}                                           /* OC = 1 */
  PUT d_i in F                                                /* OC = 1 */
  PUT F in {F(E_i)}                                           /* OC = 1 */
end for

/* OC = 1 + 1 + (3)(70 + 6 + 1 + 1 + 1 + 1) = 242 */
```

Alg. 6: Mesh Representation F1: Create face from vertices

5.3 Mesh representation F3

The schematic for mesh representation F3 is shown in Figure 4. The upward connections in this representation are from vertices directly to faces and faces to regions.

The storage for entities in this representation is $25nM_e$, as before. The storage requirements for the upward connectivity is $35nM_c$ from vertices to faces and $24nM_c$ from faces to regions. Therefore, the total storage requirement for a mesh with this type of representation is $(25M_e + 129M_c)n$.

The operation counts for the data structure are shown in Tables 2 and 4.

5.4 Mesh Representation F4

The schematic for mesh representation F4 is shown in Figure 5. Like the previous representation, it has all the entities but only two of the three upward adjacencies, namely, vertex to edge and edge to regions. The entity storage for this mesh representation is $25nM_e$. The storage for the connectivity information is $119nM_c$. The operation counts for the data structure are shown in Tables 2 and 4.

5.5 Mesh Representation F5

Figure 6 shows a schematic of mesh data representation F5. The special feature of this representation is that all the upward adjacencies originate from the vertex.

The storage required by the entities is $25nM_e$ as before since all the entities are represented. The storage required by the downward adjacencies is $70nM_c$ and that required by the upward adjacencies is $72nM_c$. Therefore, the total storage requirement for this data structure is $(25M_e + 142M_c)n$ which is almost equal to the storage requirements for the mesh representation F1.

The operation counts for the data structure are shown in Tables 2 and 4. The tables show that the operation counts for adjacency retrieval and entity creation in this representation are similar to or lesser

```
CREATE region object R                                          /* OC = 1 */
PUT R in mesh region list                                       /* OC = 1 */
for (i = 1, ..., 4) do
  GET V'1,V'2,V'3 of the i'th face using template               /* OC = 3 */
  GET Fi(V'1, V'2, V'3)                                         /* OC = 155 */
  if (Fi == 0) then
    CREATE Fi(V'1, V'2, V'3)                                    /* OC = 242 */
    ASSIGN di from template                                     /* OC = 1 */
  else
    /* Get the vertices of Fi in order */
    GET {V''1(Fi), V''2(Fi), V''3(Fi)}                          /* OC = 13 */
    GET V''k corresponding to V'1                               /* OC = 3 */
    CHECK if V''(k+1)%3 == V'2                                  /* OC = 1 */
    if (V''(k+1)%3 == V'2) then
      ASSIGN di ← 1                                             /* OC = 1 */
    else
      ASSIGN di ← 0                                             /* OC = 1 */
    end if
  end if
  PUT Fi in {F(R)}                                              /* OC = 1 */
  PUT di in R                                                   /* OC = 1 */
  PUT R in {R(Fi)}                                              /* OC = 1 */
end for

/* OC = 1 + 1 + (4)(3 + 155 + 243 + 1 + 1 + 1) = 1618 */
```

Alg. 7: Mesh Representation F1: Create region from vertices

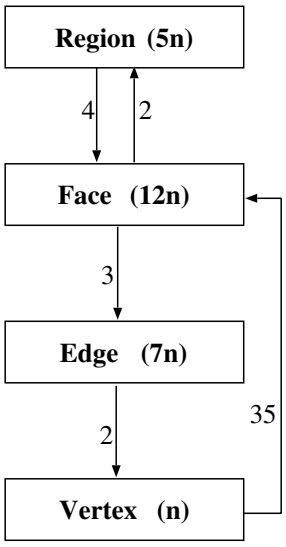

Figure 4: Mesh representation F3 with all downward adjacencies and upward adjacencies from vertices to faces and faces to regions

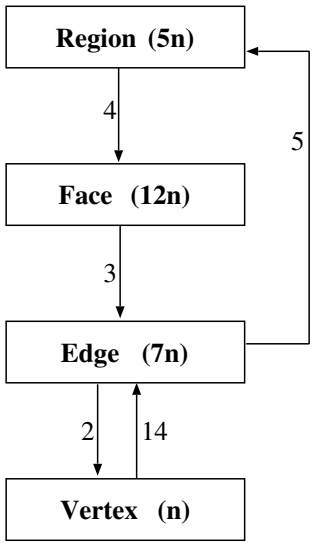

Figure 5: Mesh representation F4

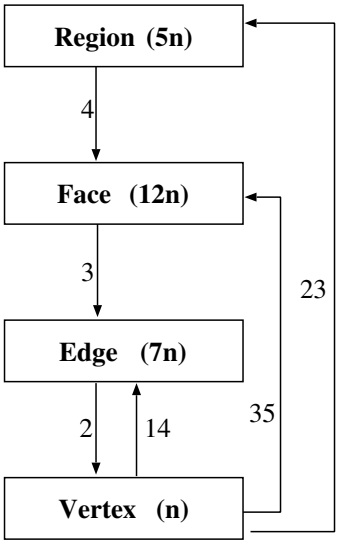

Figure 6: Mesh representation F5

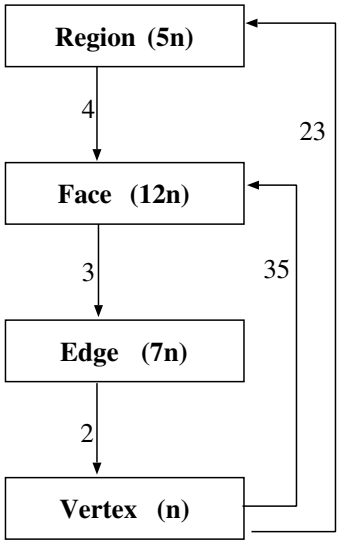

Figure 7: Mesh representation F6

than those for representation F1 at nearly the same storage cost.

5.6 Mesh Representation F6

In this representation shown in Figure 7, the upward adjacencies that are maintained are vertex-region and vertex-face. The storage requirements for this data structure are $(25M_e + 128M_c)n$. The operation counts for the data structure are shown in Tables 2 and 4

6 Reduced Mesh Representations

In this section, mesh data structures without one or more intermediate dimension topological entity types will be discussed. A mesh data structure must contain at least the lowest order entities (vertices) and the highest order entities (regions in 3D meshes and faces in 2D meshes). However, depending on the application it is possible to omit edges, faces or both from the representation. These entities may not be needed at all in the application or if they are, they may be created each time on the fly. Such types of representations are called *reduced* mesh representations. Reduced representations offer economy of storage over full representations as will be seen below. However, they may not always be complete for the purposes of an application. Computation of the operation counts for reduced mesh representations must take into account the cost of creating entities that are not explicitly represented in the data structure.

6.1 Mesh Representation R1

This is the simplest and most commonly used reduced mesh representations containing just regions and vertices with adjacency links going both ways as shown in Figure 8. In the figure, the entities and connections in dotted line are not explicitly stored. Naturally, the storage cost of this data structure is expected to be low. This representation is a refinement of the classic element-node mesh data structure.

Entities that do not exist in the representation (faces and edges) are created as required on the fly using pre-defined templates. For example, if faces do not exist in a representation they must be created as needed using templates. These templates describe the face of a region in terms of vertices of the region. Consider a

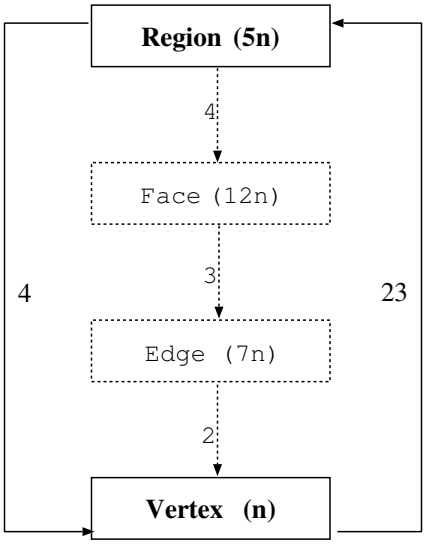

Figure 8: Mesh representation R1

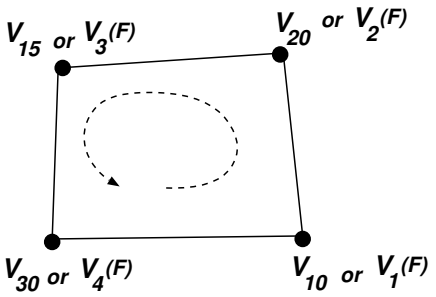

Figure 9: Face creation using templates

tetrahedron with vertices $\{V_1, V_2, V_3, V_4\}$ and the cross-product of the vector from V_1 to V_2 with the vector from V_1 to V_3 pointing into the tetrahedron. Then the templates for the faces $\{F_1, F_2, F_3, F_4\}$ are given by vertex triplets which may be $\{V_1, V_2, V_3\}$, $\{V_1, V_2, V_4\}$, $\{V_2, V_3, V_4\}$ and $\{V_3, V_1, V_4\}\}$.

One of the issues with using templates to create entities on the fly is consistency [7]. If a fixed template is used to create the faces of regions, then for two adjacent regions the direction of their common face may be inconsistent. In one possible scenario, if the face is created from the first region, it points into the first region and out of the second. If created from the second, it points out of the first region and into the second. This can lead to confusing results for topological and geometric queries to the mesh database.

To create consistently oriented entities each time, conventions may be adopted using global identifiers (*ID*s) for the component entities. For example, a convention must be adopted such that the first and the second vertices of a temporary edge remain the same during each incarnation. One way to ensure this is by creating each edge such that the vertex with the smaller ID is the first vertex of the edge. Similarly, a face must be created each time so that its orientation is the same for each instance. Therefore, the vertices from the template are reordered so that the vertex with the smallest ID is the first vertex. The second vertex is chosen such that it is an adjacent vertex with the next smallest ID. For example, assume that the template for a vertex indicates that it is made up of vertices $\{V_{10}, V_{20}, V_{15}, V_{30}\}$ as shown in Figure 9, where the vertex numbers are global IDs in the mesh. Then, V_{10} is chosen as the first vertex of the face. The vertex with the next smallest ID is V_{15} but it is not adjacent to V_{10}. Therefore, V_{20} is picked as the next vertex of the face. Picking two vertices consistently guarantees that the face is always oriented consistently.

The other issue with temporary entities in reduced representations is their comparison (Also see [12]). In full representations, it is possible to check if two entities are the same just by comparing their pointers, or addresses or IDs. In reduced representations, if the same edge is created twice then the two instances of the edge will have different address or pointers. One way of checking if two objects are instances of the same edge is to check if their vertices are the same (assuming mesh representations with linear entities only). Similarly, two objects can be checked if they are instances of the same face by checking their vertices. However, the operation count for such a verification for an edge is 4. For verifying that two objects refer to the same triangular face, the operation count is 9. It would be desirable to have a more economical system of comparing entities.

One possibility is to assign the same unique ID number to any instance of an edge or a face. For instance, the ID number of an edge may be uniquely derived from the IDs of its vertices. This can be done by left

shifting the ID of one of the vertices and doing a binary OR with the ID of the other vertex. Then the IDs of multiple instances of the edge can be compared and determined to be of the same edge.

Another possibility is that when an edge object is created it is never destroyed (unless the algorithm calls for the removal of the edge from the mesh). Instead it is placed in a database (e.g. hash table) from which it can be easily accessed, given its component vertices. When the edge is called upon at a later time, the same object is retrieved instead of creating a new instance of the edge.

The first solution is computationally efficient *if* unique IDs can be derived for every entity. The worst case is when unique identifiers for a quad face must be derived as a function of four vertex IDs. Since an identifier has to be a finite length integer this places restrictions on the highest ID number that a vertex can have and thereby on the largest size mesh that can be handled. If it is assumed that the largest integer that can be handled is a 64-bit unsigned integer, each vertex ID cannot occupy more than 16 bits for a quad mesh. In other words, the largest vertex ID can be 65535 which is sorely inadequate for today's large simulations. The second solution does not have such stringent restrictions on the size of the mesh. However, it can be more costly both in terms of computation and memory since entities have to be retrieved from a hash table. Also, in the worst case, one might access every edge and face in the mesh in which case all of them are present simultaneously in the hash table. The advantage of a reduced representation is nullified in such a case.

For this discussion it is assumed that the first solution is used and that the cost of comparing two entities temporary or permanent is 1.

6.1.1 Storage:

The storage cost for the entities ($5n$ regions and n vertices) is ($6nM_e$) in representation R1. The cost for storing adjacency information is $(5n)(4)M_c + (n)(23)M_c = 43nM_c$. Therefore, the total cost of storage for the data structure is $(6M_e + 43M_c)n$.

6.1.2 Operation Counts:

1. $\{F(R)\}$: $OC = 72$, See Alg. 8.

   Faces do not exist in this representation so they must be created as needed using templates. These templates describe the faces of a region in terms of its vertices. For a tetrahedron with vertices $\{V_1, V_2, V_3, V_4\}$ and the cross-product of the vector from V_1 to V_2 with the vector from V_1 to V_3 points into the tetrahedron. Then the templates for the faces $\{F_1, F_2, F_3, F_4\}$ are given by the vertex triplets $\{V_1, V_2, V_3\}$, $\{V_1, V_4, V_2\}$, $\{V_2, V_4, V_2\}$ and $\{V_3, V_4, V_1\}$. The ordering of the vertices in each face will be such that consistency is ensured.

2. $\{E(R)\}$: $OC = 58$, See Alg. 9

   Edges don't exist in this representation either and must be created on the fly. The edge template for a tetrahedral region is $\{V_1, V_2\}$, $\{V_2, V_3\}$, $\{V_3, V_1\}$, $\{V_1, V_4\}$, $\{V_2, V_4\}$, $\{V_3, V_4\}$.

3. $\{V(R)\}$: $OC = 4$, as vertices are directly referenced by the regions.

4. $\{R(F)\}$: $OC = 302$.

   For this operator, it must be recognized that F is a temporary face defined in terms of its vertices. Therefore, the algorithm first accesses a vertex of the face, gets the regions connected to that vertex and picks the regions that also contain the other vertices of the face.

5. $\{E(F)\}$: $OC = 24$; the edges of the face, which are defined in terms of its vertices, are retrieved similar to the way they are retrieved for regions.

6. $\{V(F)\}$: $OC = 3$, as faces are defined directly in terms of vertices.

7. $\{R(E)\}$: $OC = 214$, similar to the algorithm for $\{R(F)\}$.

```
GET {V(R)}                                                    /* OC = 4 */
for (F_i(R), i = 1, ..., 4) do
  GET vertices {V_j(F_i), j = 1, 2, 3} from template          /* OC = 3 */
  GET vertex V_{k_1} with smallest ID                         /* OC = 3 */
  ASSIGN V'_1 ← V_{k_1}                                       /* OC = 1 */
  GET vertex V_{k_2} with second smallest ID                  /* OC = 3 */
  ASSIGN V'_2 ← V_{k_2}                                       /* OC = 1 */
  CHECK if k_1 < k_2                                          /* OC = 3 */
  if (k_1 < k_2) then
    ASSIGN V'_3 ← V_{(k_2+1)%3}                               /* OC = 1 */
  else
    ASSIGN V'_3 ← V_{(k_1+1)%3}                               /* OC = 1 */
  end if
  CREATE new face object for F_i                              /* OC = 1 */
  PUT links to {V'_l, l = 1, 2, 3} in F_i                     /* OC = 3 */
  PUT F_i in {F(R)}                                           /* OC = 1 */
end for

/* OC = 4 + (4)(3 + 3 + 1 + 3 + 1 + 1 + 1 + 3 + 1) = 72 */
```

Alg. 8: Mesh Representation R1: $\{F(R)\}$

```
GET {V(R)}                                                    /* OC = 4 */
for (E_i(R), i = 1, ..., 6) do
  GET vertices of {V_j(E_i), j = 1, 2} from template          /* OC = 2 */
  GET vertex V_{k_1} with smallest ID                         /* OC = 1 */
  ASSIGN V'_1 ← V_{k_1}                                       /* OC = 1 */
  ASSIGN V'_2 ← V_{(k_1+1)%2}                                 /* OC = 1 */
  CREATE new edge object for E_i                              /* OC = 1 */
  PUT links to {V'_1, V'_2} in E_i                            /* OC = 2 */
  PUT E_i in {E(R)}                                           /* OC = 1 */
end for

/* OC = 4 + (6)(2 + 1 + 1 + 1 + 1 + 2 + 1) = 58 */
```

Alg. 9: Mesh representation R1: $\{E(R)\}$

8. $\{F(E)\}$: $OC = 721$.

   The algorithm for this operator accesses the regions connected to one vertex of the edge, gets the vertex templates for the faces of each of the regions, checks if the other vertex is also in the template for the face and then constructs the face. To avoid duplication from two neighboring regions, the procedure must also perform a check before adding the dynamically created face to the list of faces connected to the edge.

9. $\{V(E)\}$: $OC = 2$.

10. $\{R(V)\}$: $OC = 23$, as vertices are directly connected to regions using them.

11. $\{F(V)\}$: $OC = 3462$, similar to the algorithm for $\{F(E)\}$.

12. $\{E(V)\}$: $OC = 1969$, similar to the algorithm for $\{F(E)\}$.

13. Create V: $OC = 5$.

14. Create $E(V_1, V_2)$: $OC = 5$; the procedure for this is similar to that of the full representations except that the vertex with the lower ID must always be added as the first vertex of the edge to ensure consistency.

15. Create $F(E_1, E_2, E_3)$: $OC = 21$; in this procedure, the vertices of the face have to be derived from the ordered set of edges and assigned to the face object. The computational complexity is therefore similar to the full representations where the same had to be done to determine the direction in which the face uses the edges.

16. Create $F(V_1, V_2, V_3)$: $OC = 11$; this procedure is almost a direct assignment of the vertices to the face object except that the vertices must first be ordered such that the vertex with the lowest ID is the first vertex and the second vertex is an adjacent vertex with the lowest possible ID.

17. Create $R(F_1, F_2, F_3, F_4)$: $OC = 25$; this procedure involves finding the vertices of the faces and assigning them to the region object in a specific manner.

18. Create $R(V_1, V_2, V_3, V_4)$: $OC = 10$, as this is a direct assignment of the vertices to the regions. Note that in this process, the region is also assigned to the region sets of the vertices.

The operation counts for the data structure are summarized in Figure 8 and in Tables 2 and 4

6.2 Reduced Mesh Representation R2

This data structure is one of the many possible variations of the R1 data structure (also see [1, 2, 3, 14]). Only regions and vertices are explicitly represented in this data structure with both region-vertex and vertex-regions links (as in mesh representation R1). However, additional information is stored with regions and vertices to make some adjacency queries more economical. Regions in this representation store information about their adjacent regions (other regions with which they share a virtual face). Vertices also store information about their adjacent vertices, i.e., vertices they are connected to by a virtual edge. The mesh representation is shown in Figure 10 where the looped back connections indicate a connection to adjacent entities of the same type. The storage requirements for the entities for this data structure are $(6M_e + 43M_c)n$ as in representation R1 plus an additional amount due to storage of the extra links. Each tetrahedron has 4 faces through which it connects to 4 other regions and therefore, an addition $(5n)(4M_c)$ is added to the storage. Each vertex is connected to an average of 14 "edges"; in other words, it is adjacent to an average of 14 other vertices. This adds $(n)(14M_c)$ to the storage of this data structure. Therefore, the total storage is $(6M_e + 77M_c)n$.

The operation counts for the adjacency and creation operators are summarized in Tables 2 and 4. The additional links stored in this data structure reduce the cost of upward adjacency queries $R(F)$ and $E(V)$.

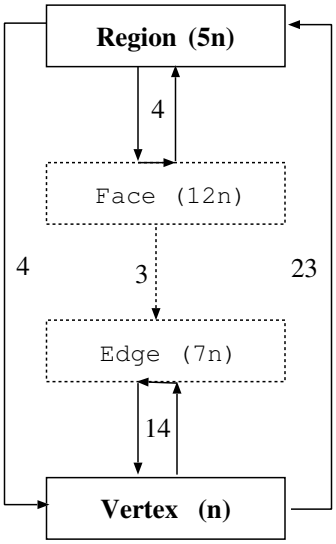

Figure 10: Reduced mesh representation R2

However, region creation becomes more expensive than in representation R1 since the additional connectivity information has to be derived and incorporated into the data structure for later use. In this data structure, unlike the other reduced data structures, it is advantageous to incorporate upward adjacency information in temporary faces. This is due to the easily accessible region-region connectivity information. Therefore, the face retrieval operators are slightly more expensive than the corresponding ones of representation R1.

6.3 Reduced Mesh Representation R3

This data structure considers a reduced representation where edges are not explicitly represented but vertices, faces and regions are. The motivations for representing faces as entities are:

- Inclusion of faces as real entities allows unambiguous representation of the most common types of non-manifold models (combinations of regions and faces).
- Temporary faces are more complex to create than temporary edges.

The connections chosen for the data structure are 1-level upward and downward adjacencies between entities as shown in Figure 11. The storage for this data structure is more than the minimal reduced representation but the computational cost is expected to be better. The regions in this representation are defined by their faces and the faces are defined by their vertices.

The storage requirements for the entities for this data structure are $5nM_e + 12nM_e + nM_e = 18nM_e$. The connections require $((5n)(4) + (12n)(3) + (n)(35) + (12n)(2))M_c = 115nM_c$.

The operation counts for the data structure are shown in Tables 2 and 4

6.4 Reduced Mesh Representation R4

This data structure considers a variation of representation R3 reduced representation in which an aspect of the connectivity is borrowed from representation R2. Therefore, in addition to the entities and links present in representation R3, vertices also contain information about their neighboring vertices. The data structure is shown in Figure 12. The computational costs for adjacency retrieval are similar to that of representation R3. The computational costs for entity creation are similar to that of representation R2.

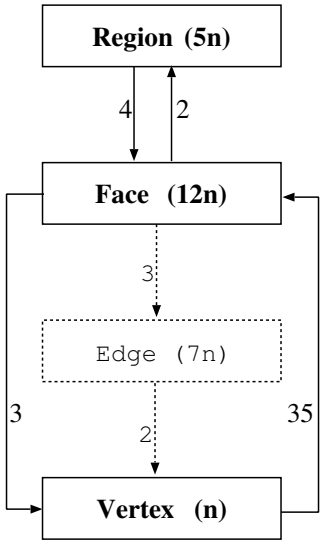

Figure 11: Reduced mesh representation R3

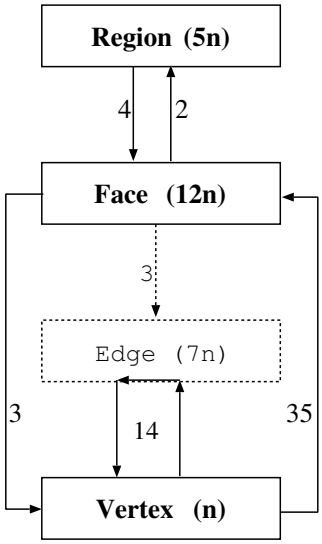

Figure 12: Reduced mesh representation R4

The storage requirements for the entities for this data structure are $5nM_e + 12nM_e + nM_e = 18nM_e$. The connections require $((5n)(4) + (12n)(3) + (n)(35) + (12n)(2) + (14n)(1))M_c = 129nM_c$.

The operation counts for the data structure are shown in Tables 2 and 4

7 Summary

Given below in Table 2 are the operation counts for adjacency operators for the various mesh representations. These calculations have been presented in the previous sections and made use of entity marking wherever local searching was necessary. Also, the algorithms for the different operators are given in detail in [13].

In Table 4, the cost of creating entities is summarized for each mesh representation. In this table, the cost of creating faces and regions in two different ways is presented, one from the next lower order entities (edges for face creation and faces for region creation) and the other from the lowest order entities, namely, vertices. When faces and regions are created from vertex information alone, then the creation procedures must perform local searching to identify intermediate order entities (e.g. edges). Therefore, they tend to be more expensive than creation procedures which are given information about next lower order entities directly, particularly in the full representations. The cost of intermediate creation procedures such as the creation of a tetrahedron from a triangle and a fourth vertex is expected to be in between the two extrema. This is evident in the table, except in the case of representation R1 where the regions are represented directly in terms of vertices.

From the summary data given above, there is no optimal data structure that minimizes storage as well as computational cost for all operators. Also, applications may invoke some operators more than others and not invoke other operators at all making it difficult to derive a computational cost valid for all applications. Therefore, it is possible to draw only general conclusions about the advantages and drawbacks of the data structures for use in all applications. For any one application it is possible to be more specific and quantitative about the relative performance of the data structures.

The method used here to calculate the computational efficiency of a data structures for a particular application is as follows:

- The average number of times each operator is called (relative to the count of any one operator) is

Table 2: Adjacency retrieval operation counts for tetrahedra for the ten mesh representations. Also shown are the storage requirements per node of the data structures.

Type	Size/n	$F(R)$	$E(R)$	$V(R)$	$R(F)$	$E(F)$	$V(F)$	$R(E)$	$F(E)$	$V(E)$	$R(V)$	$F(V)$	$E(V)$
F1	$25M_e + 143M_c$	4	36	30	2	3	13	50	5	2	619	399	14
F2	$25M_e + 93M_c$	4	36	30	211	3	13	512	877	2	23	1394	1749
F3	$25M_e + 129M_c$	4	36	30	2	3	13	297	252	2	360	35	840
F4	$25M_e + 119M_c$	4	36	30	48	3	13	5	210	2	339	1710	14
F5	$25M_e + 142M_c$	4	36	30	211	3	13	512	252	2	23	35	14
F6	$25M_e + 128M_c$	4	36	30	211	3	13	512	252	2	23	35	840
R1	$6M_e + 43M_c$	72	58	4	302	24	3	214	721	2	23	3462	1969
R2	$6M_e + 77M_c$	84	58	4	2	24	3	214	731	2	23	3532	84
R3	$18M_e + 115M_c$	4	58	21	2	24	3	267	251	2	360	35	1449
R4	$18M_e + 129M_c$	4	58	21	2	24	3	267	251	2	360	35	84

Table 3: Average relative call frequency of adjacency retrieval from the MEGA software from SCOREC, RPI [15].

$F(R)$	$E(R)$	$V(R)$	$R(F)$	$E(F)$	$V(F)$	$R(E)$	$F(E)$	$V(E)$	$R(V)$	$F(V)$	$E(V)$
73	23	429	543	8121	185	1	950	12006	27	48	2102

Table 4: Operation counts of entity creation procedures for tetrahedra using the ten different mesh representations.

Type	V_{new}	$E_{new}(V_{1,2})$	$F_{new}(V_{i=1,\ldots,3})$	$F_{new}(E_{i=1,\ldots,3})$	$R_{new}(V_{i=1,\ldots,4})$	$R_{new}(F_{i=1,\ldots,4})$
F1	5	6	242	24	1618	14
F2	5	4	5060	21	13918	48
F3	5	4	2714	24	6726	14
F4	5	6	239	21	1622	56
F5	5	6	242	24	1782	48
F6	5	4	2714	24	6726	48
R1	5	5	11	21	10	25
R2	5	5	11	21	1384	103
R3	5	5	8	17	1462	14
R4	5	5	8	17	1528	90

Table 5: Relative call frequency for entity creation for tetrahedra (estimated using relative numbers of entities in a typical tetrahedral mesh).

Type	V_{new}	$E_{new}(V_{1,2})$	$F_{new}(V_{i=1,\ldots,3})$	$F_{new}(E_{i=1,\ldots,3})$	$R_{new}(V_{i=1,\ldots,4})$	$R_{new}(F_{i=1,\ldots,4})$
Ave.	1	7	12/2	12/2	5/2	5/2

found.

- The cost of invoking each operator is multiplied with its relative call frequency to produce a weighted cost for the application.
- The weighted costs of calling the operators are summed up to give an overall computational cost for the particular application.

This gives a quantitative idea of the relative merits of different data structures for the particular application. To illustrate this process, a well developed meshing environment (MEGA, developed by the Scientific Computation Research Center (SCOREC), Rensselaer Polytechnic Institute(RPI) [15]) was used to generate solid meshes of 19 geometric models while exercising its various capabilities like quality improvement, boundary layer creation etc. The relative call frequencies of the various adjacency operators, normalized with respect to the number of calls to the operator $R(E)$ is shown in Table 3 for the test cases. The average of these relative call frequencies is also shown in the table.

Using the average call frequency for the adjacency operators, the weighted cost of each operator for the different types of representations is computed and the mesh representations ranked. The mesh representations in increasing order of the computational cost of adjacency retrieval using this measure are F1, F4, F5, R4, R2, F3, F6, R3, F2, R1. The ranking of the mesh data structures with respect to the computational cost of adjacency retrieval will be denoted R_a.

The relative call frequency of the creation operators is an assumed quantity based on the relative numbers of entities in a typical mesh. For entities that have only one type of creation operator, like vertices and edges, the relative call frequency is assumed to be the number of entities of that type relative to the number of vertices in a typical mesh. For higher order entities with multiple creation methods, like faces and regions, it is assumed that an equal number of the entities are created using the least expensive and the most expensive method. For example, the number of regions in a typical tetrahedral mesh is 5 times the number of vertices. Therefore, it is assumed that the relative call frequencies of the least and most expensive region creation operators are 2.5 each. The assumed relative call frequency of the various creation operators (with respect to the number of vertices in the mesh) for a tetrahedral mesh is given in Table 7. Multiplying the costs of the creation operators with their relative call frequencies and summing up for each data structure gives an estimate of the cost of creating entities in each data structure. Using this measure, the various data structures in increasing order of the cost of entity creation is R1, R3, R2, R4, F1, F4, F5, F3, F6, F2. The ranking of the mesh data structures with respect to the computational cost of creation will be denoted R_c.

In calculations of the storage cost, it is observed that for any value of M_e greater than $3.15M_c$, the ranking of the mesh data structures does not change. Therefore, it is assumed (somewhat arbitrarily) that $M_e = 5$ and $M_c = 1$. The different mesh representations in increasing order of the resulting storage cost are R1, R2, R3, F2, R4, F4, F6, F3, F5, F1. The ranking of the mesh data structures with respect to the storage cost will be denoted R_s.

The best data structure for mesh generation is the one that has a low storage cost and a low computational cost for adjacency retrieval and entity creation. For other applications in which the mesh is static (like some analysis programs), the storage and adjacency retrieval costs are the important factors. Therefore, an

Table 6: Individual and overall rank of different mesh representations for tetrahedral meshes. R_s is the ranking in ascending order of storage requirements, R_a is the ranking in ascending order of computational cost of adjacency retrieval and R_c is the ranking in ascending order of computational cost of entity creation.

Type	R_s	R_a	R_c	$R_1 = \sqrt{R_s^2 + R_a^2 + R_c^2}$	$R_2 = \sqrt{R_s^2 + R_a^2}$
F1	10	1	5	11.22 (6)	10.05 (9)
F2	4	9	10	14.04 (10)	9.85 (6)
F3	8	6	8	12.81 (8)	10.00 (8)
F4	6	2	6	8.72 (3)	6.32 (2)
F5	9	3	7	12.22 (7)	9.49 (5)
F6	7	7	9	13.38 (9)	9.90 (7)
R1	1	10	1	10.10 (5)	10.05 (9)
R2	2	5	3	6.16 (1)	5.39 (1)
R3	3	8	2	8.77 (4)	8.54 (4)
R4	5	4	4	7.55 (2)	6.40 (3)

overall rank, R_1, is devised for each of the mesh representations using a root mean squared value of the storage ranking, R_s, and the two efficiency rankings, R_a and R_c. Thus, the overall RMS rank is given as $R = \sqrt{R_s^2 + R_a^2 + R_c^2}$. Based on this measure, the ranking of the various representations is R2, R4, F4, R3, R1, F1, F5, F3, F6, F2 as shown in Table 6. In the table, values of an alternate rank, R_2, based only on R_s and R_a are also presented for the different data structures.

Other possibilities exist for computing an overall rank, such as the sum of the storage and efficiency ranks, or the sum of the absolute values of the deviation of the ranks from the mean ranks, etc. However, the root mean square method gives the best results for a data structure that is both compact and efficient. Also, representations R2, R4, F4 are consistently among the top performers for tetrahedral meshes using some of the other measures as well.

Table 7 shows the operation counts for the various adjacency operators for a hexahedral mesh. As expected many of the upward adjacency operators are less expensive than in a tetrahedral mesh while the downward adjacency operators are more expensive. This is because each hexahedron has more lower order entities than a tetrahedron and each lower order entity in a hexahedral mesh is connected to fewer higher order entities than in a tetrahedral mesh. The weighting factors used to calculate the overall computational cost for hexahedral meshes are taken from the tetrahedral mesh generator case due to the lack of such data for a hexahedral mesh generator The cost of the entity creation operators, the assumed call frequencies of the various operators and their weighted cost according to the principles used above for tetrahedral meshes are given in Tables 8, 9.

Table 10 summarizes the ranks of the different representations for hexahedral meshes in the three categories. According to the root mean square ranking proposed for the tetrahedral meshes, the overall ranking of the data structures for hexahedral meshes is R4, R2, R3, R1, F4, F3, F1, F5, F6, F2.

The results for other meshes such as mixed meshes with tetrahedra, hexahedra and triangular prisms are expected to be similar but have not been examined.

From the above discussion, it is clear that representations R2, R4 and F4 are good data structures for the mesh generation program used as a test case. Limited tests with a higher order finite element method for analysis of acoustics problems [16] has also provided similar indication. It is expected that the rankings presented will vary with the mesh generation and analysis algorithms used as the weights will tend to be different. However, it can be said that the data structures R2, R4 and F4 offer a good compromise in terms of storage versus computational cost. If a choice of only one data structure for mesh representation for all types of meshes and for mesh generation and analysis algorithms were to be made then either of them would be a

Table 7: Operation Counts for Adjacency Retrieval for hexahedra for the ten different mesh representations. Also shown are the storage requirements of the data structure per node.

Type	Size/n	$F(R)$	$E(R)$	$V(R)$	$R(F)$	$E(F)$	$V(F)$	$R(E)$	$F(E)$	$V(E)$	$R(V)$	$F(V)$	$E(V)$
F1	$8M_e + 48M_c$	6	96	22	2	4	10	32	4	2	272	138	6
F2	$8M_e + 32M_c$	6	96	22	108	4	10	321	557	2	8	692	1382
F3	$8M_e + 42M_c$	6	96	22	2	4	10	142	114	2	124	12	382
F4	$8M_e + 42M_c$	6	96	22	55	4	10	4	288	2	118	790	6
F5	$8M_e + 50M_c$	6	96	22	108	4	10	321	114	2	8	12	6
F6	$8M_e + 44M_c$	6	96	22	108	4	10	321	114	2	8	12	382
R1	$2M_e + 16M_c$	134	116	8	238	32	4	102	827	2	8	1268	974
R2	$2M_e + 28M_c$	152	116	8	2	32	4	102	835	2	8	1292	36
R3	$5M_e + 36M_c$	6	116	18	2	32	4	122	4	2	124	12	498
R4	$5M_e + 42M_c$	6	116	18	2	32	4	122	4	2	124	12	36

Table 8: Operation counts of entity creation procedures for tetrahedra using the ten different mesh representations.

Type	V_{new}	$E_{new}(V_{1,2})$	$F_{new}(V_{i=1,\ldots,4})$	$F_{new}(E_{i=1,\ldots,4})$	$R_{new}(V_{i=1,\ldots,8})$	$R_{new}(F_{i=1,\ldots,6})$
F1	5	6	322	30	2570	20
F2	5	4	5438	26	20332	50
F3	5	4	1658	30	5870	20
F4	5	6	158	26	2870	128
F5	5	6	322	30	1852	50
F6	5	4	1658	30	5872	50
R1	5	5	13	28	18	26
R2	5	5	13	28	1770	123
R3	5	5	10	25	1334	20
R3	5	5	10	25	1418	120

Table 9: Relative call frequency for entity creation for hexahedra (estimated using relative numbers of entities in a typical hexahedral mesh).

Type	V_{new}	$E_{new}(V_{1,2})$	$F_{new}(V_{i=1,\ldots,4})$	$F_{new}(E_{i=1,\ldots,4})$	$R_{new}(V_{i=1,\ldots,8})$	$R_{new}(F_{i=1,\ldots,6})$
Ave.	1	3	3/2	3/2	1/2	1/2

Table 10: Individual and overall rank of different mesh representations for hexahedral meshes. R_s is the ranking in ascending order of storage requirements, R_a is the ranking in ascending order of computational cost of adjacency retrieval and R_c is the ranking in ascending order of computational cost of entity creation.

Type	R_s	R_a	R_c	$R_1\sqrt{R_s^2+R_a^2+R_c^2}$	$R_2\sqrt{R_s^2+R_a^2}$
F1	9	1	7	11.58 (7)	9.22 (7)
F2	5	10	10	15.00 (10)	11.18 (10)
F3	6	5	8	11.18 (6)	7.81 (4)
F4	6	4	6	9.38 (5)	7.21 (2)
F5	10	2	5	11.83 (8)	10.02 (9)
F6	8	6	9	13.45 (9)	10.00 (8)
R1	1	9	1	9.11 (4)	9.06 (6)
R2	2	7	4	8.31 (2)	7.28 (3)
R3	3	8	2	8.77 (3)	8.54 (5)
R4	4	3	3	5.83 (1)	5.00 (1)

good choice. Between the three, one may decide on the full data structure, F4 if it is necessary to represent faces and edges and derive classifications for them. If explicit representation and unique classification is necessary only for faces, the reduced representation R4 may be used and representation R2 may be used otherwise.

8 Conclusion

The issue of mesh data structure selection for mesh generation and analysis applications has been addressed in this article. An appropriate choice of mesh representation is central to minimizing computational and storage costs in an application. A methodology for analyzing the storage and computational costs of different data structures was presented. Since no single representation can satisfactorily meet all requirements, a measure of goodness was devised for evaluating mesh data structures. Using the proposed tools, a number of different mesh data structures were analyzed. Results for 10 of these data structures, 6 full and 4 reduced, were presented. It was seen that representation F4 consistently performed the best among all full representations for tetrahedral and hexahedral meshes for mesh generation and analysis. Among reduced representations, R2 and R4 were the top performers. Even though the ranking is specific for the applications used to collect call frequency data, the methods and measures presented are general enough to analyze any mesh data structure for any given application.

References

[1] LaGriT - Los Alamos Grid Toolbox. Technical report, Los Alamos National Laboratory, Los Alamos, NM, 1995. (`http://www.t12.lanl.gov/home/lagrit/`).

[2] N. A. Golias and T. D. Tsiboukis. An approach to refining three-dimensional tetrahedral meshes based on delaunay transformations. *International Journal of Numerical Methods in Engineering*, 37:793–812, 1994.

[3] N. P. Weatherill and O. Hassan. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering*, 37:2005–2039, 1994.

[4] T. J. Barth. Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes equations. *Int. J. Engineering Science*, 25(6):623–653, 1987.

[5] W. J. Schroeder and M. S. Shephard. Geometry-based fully automatic mesh generation and the Delaunay triangulation. *International Journal for Numerical Methods in Engineering*, 26:2503–2515, 1988.

[6] M. S. Shephard and M. K. Georges. Automatic three-dimensional mesh generation by the Finite Octree technique. *International Journal for Numerical Methods in Engineering*, 32(4):709–749, 1991.

[7] M. W. Beall and M. S. Shephard. A general topology-based mesh data structure. *International Journal for Numerical Methods in Engineering*, 40(9):1573–1596, May 1997.

[8] M. Mäntylä. *Introduction to Solid Modeling*. Computer Science Press, Rockville, Maryland, 1988.

[9] K. J. Weiler. The radial-edge structure: A topological representation for non-manifold geometric boundary representations. In M. J. Wozny, H. W. McLaughlin, and J. L. Encarnacao, editors, *Geometric Modeling for CAD Applications*, pages 3–36. North Holland, 1988.

[10] M. S. Shephard, S. Dey, and M. K. Georges. Automatic meshing of curved three-dimensional domains: Curving finite elements and curvature-based refinement. In I. Babuska, J. E. Flaherty, J. E. Hopcroft, W. D. Henshaw, J. E. Oliger, and T. Tezduyar, editors, *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, volume 75, pages 67–96. Springer-Verlag, 1995. IMA Volumes in Mathematics and its Applications.

[11] S. Dey, M. S. Shephard, and J. E. Flaherty. Geometry representation issues associated with p-version finite element computations. *Computer Methods in Applied Mechanics and Engineering*, 150(1-4):39–55, 1997.

[12] J.-F. Remacle, B. K. Karamete, and M. S. Shephard. Algorithm oriented mesh database. In *Proceedings of the 9th International Meshing Roundtable*, pages 349–359, New Orleans, LA, October 2000. Sandia National Laboratories.

[13] R. V. Garimella and C. W. Gable. Split finite volumes for the solution of the transient diffusion equation in heterogeneous material models. *J. of Computational Physics*, 2001. Under review.

[14] M. J. Marchant and N. P. Weatherill. Adaptivity techniques for compressible inviscid flows. *Computer Methods in Applied Mechanics and Engineering*, 106:83–106, 1993.

[15] M. S. Shephard. Meshing environment for geometry based analysis. *International Journal of Numerical Methods in Engineering*, 47:169–190, 2000.

[16] S. Dey, J.J. Shirron, and L.S. Couchman. Mid-frequency stuctural acoustic and vibration analysis in arbitrary, curved three-dimensional domains. *Computers and Structures*, 79(6):617–629, Jan 2001.